



## Research Article

# Secure Storage and Retrieval of IoT Data Based on Private Information Retrieval

Khaled Riad <sup>1,2</sup> and Lishan Ke <sup>3</sup>

<sup>1</sup>School of Computer Science, Guangzhou University, Guangzhou 510006, China

<sup>2</sup>Mathematics Department, Faculty of Science, Zagazig University, Zagazig 44519, Egypt

<sup>3</sup>College of Mathematics and Information Science, Guangzhou University, Guangzhou 510006, China

Correspondence should be addressed to Khaled Riad; [khaled.riad@science.zu.edu.eg](mailto:khaled.riad@science.zu.edu.eg) and Lishan Ke; [kelishan@gzhu.edu.cn](mailto:kelishan@gzhu.edu.cn)

Received 25 May 2018; Accepted 29 August 2018; Published 18 November 2018

Guest Editor: Karl Andersson

Copyright © 2018 Khaled Riad and Lishan Ke. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The fast growth of Internet-of-Things (IoT) strategies has actually presented the generation of huge quantities of information. There should exist a method to conveniently gather, save, refine, and also provide such information. On the other hand, IoT data is sensitive and private information; it must not be available to potential attackers. We propose a robust scheme to guarantee both secure IoT data storage and retrieval from the untrusted cloud servers. The proposed scheme is based on Private Information Retrieval (PIR). It stores the data onto different servers and retrieves the requested data slice without disclosing its identity. In our scheme, the information is encrypted before sending to the cloud servers. It is also divided into slices of a specific size class. The experimental analysis on many different configurations supported efficiency and the efficacy of the proposed scheme, which demonstrated compatibility and exceptional performance.

## 1. Introduction

With the huge revolution of Internet-of-Things (IoT) and cloud computing as its storage environment, the user requests a query to a part of information and should receive that part without disclosing its identity. A great number of researches have been dedicated to defend the database from *curious* users. There are approaches that enable questions to be asked by an individual into a database by reconstructing the worth of entities in a manner that prevents him. If the user would like to maintain his privacy (in the information-theoretic sense), then he could request a copy of the entire database. This can cause a huge communication overhead, making it unacceptable.

Before going further let us make the problem more tangible. Let a binary string  $x = x_1, \dots, x_n$  of length  $n$ .  $k \geq 2$  server stores copies of this binary string. The user has some indicator  $i \leq n$  and he is interested in getting this little  $x_i$ . To attain this aim, the little  $x_i$  could be calculated, the user queries each of the servers and receives responses. The query to each server is distributed separately of  $i$  and each server

gains no information about  $i$ . A strategy with these properties is called a Private Information Retrieval (PIR) [1, 2].

Within this paper we introduce encrypted PIR that offers a great privacy. That is, unbounded servers should not obtain any information about the requested piece of information. One does require at least two servers to achieve privacy. These servers do not need to store the whole database; they could store portions of it. We show that when those pieces are encrypted, instead of duplicated, the memory overhead can be decreased.

*1.1. Motivation.* In addition to reducing the storage overhead caused by replicating the data to reduce the communication cost in the traditional PIR protocols [2] and achieving the information-theoretic privacy, in big data, the user who reconstructs data is distinct from the user who distributes them. Also, the user who distributes data should encrypt it using different keys and distribute the ciphertext.

Moreover, querying information over big data where no one can get the identity of the parts you are querying or the responses obtained is a big challenge and sounds like science

fiction. But it is actually the PIR science. In modern data storage systems, data are usually stored at multiple storage nodes in the cloud. The privacy of information retrieval has to be shielded. One naive method to attain PIR is by simply downloading each document in the system regardless of the user requirements. The drawback of that strategy is the very large restoration cost, which further increases with  $N$  (the range of saved documents). Thus, there is an urgent requirement to present a strong PIR strategy for storage and recovery.

*1.2. Contributions.* One of the main contributions in this paper is integrating PIR [1] with cloud computing, to guarantee the efficiency and security of encrypted storage and retrieval of information for big data queries from the untrusted cloud servers. Our scheme first divides the data into slices, then encrypts each of them, and stores those encrypted slices using the Swift service on the untrusted cloud servers. In the reconstruction stage, the requested data slice is reconstructed without disclosing any information about that requested slice, with retrieval cost independent of the number of stored slices. The main contributions can be summarized as follows:

- (i) We have integrated the PIR scheme with cloud computing, to securely store the personal information as well as efficiently and smoothly retrieve it.
- (ii) We have implemented our scheme on the top of our private cloud environment, by initializing a set of virtual instances that are divided into three categories based on their configuration and their role in the proposed scheme.
- (iii) We have tested the proposed scheme under two different scenarios, client situation and overlay situation.
- (iv) The experimental results have suggested that the decryption and retrieval cost are separate from the amount of saved slices and harmonious with all reasonable scenarios for applying our proposed scheme.

Ultimately, the throughput for a workload caused by the implementation of mixing `get` and also `put_fragment` requests on Swift is sensible and accepted by the operator and user.

*1.3. Organization.* The rest of this paper is organized as follows: Section 2 presents the related work and smooth comparison between the currently proposed information hiding and extraction approaches while not disclosing the queried information. Section 3 presents our proposed  $k$ -private and  $t$ -out-of- $n$  PIR scheme and its three stages. The proposed scheme implementation is introduced in Section 4. The comprehensive performance analysis is presented in Section 5. This is followed by the conclusion in Section 6.

## 2. Related Work and Discussion

The notion of earning the extraction of this data content of a huge data source should consider quite rough security requirements to be able to do not disclose the queried data.

Reconstructing a part of discerning information from an encrypted source is determined by the availability of the whole source, which is introduced by Rivest [3], by proposing *all-or-nothing transform* (AONT). In this scheme any modification on the encrypted message limits the ability to decrypt the resource. Thus, the AONT scheme works well for the scenarios where the user who wants to decrypt the resource has never accessed the key before. This is not realistic, because the cloud users frequently access the resources and request to decrypt their ciphertext. In our scheme, the user can decrypt the resource as long as he has the appropriate key and the sufficient data slices that can generate the requested resource. Moreover, the requesting user can only decrypt the ciphertext if he has successfully generated the decryption token. Another direction for securely uploading the data to cloud is introduced in [4], which ensures that the cloud validates the data integrity while avoiding malicious home gateways that monitor and modify the data. In our scheme the data is not stored as one block, but it is sliced into several slices based on its size. Then, it is first hidden using a permutation hash function. After that, those data slices are encrypted using an encryption algorithm and an access structure that is implicitly included in the ciphertext. In this manner we are reducing the storage time overhead. The authors in [5] formalize the notion of verifiable database with incremental updates (Inc-VDB). As well as in [6] pointing out to Catalano-Fiore's VDB framework from vector commitment is vulnerable to the so-called forward automatic update (FAU) attack.

For information hiding and securely extracting that information, there are multiple contributions based on different schemes, such as [7]. The authors introduced a T-private PIR which is a generalization of PIR to include the requirement that even if any T of the N databases colludes, the identity of the retrieved message remains completely unknown to them. But in our scheme whatever was the number of colluding databases, the user will only receive the encrypted data slices that are only sufficient for extracting the required information not more. Also, the authors in [8] utilized a new cryptographic primitive, called conditional disclosure of secrets, which we believe that it may be a useful building block for the design of other cryptographic protocols. In our scheme, we have considered slicing the data into multiple slices before encrypting and storing it on the cloud storage servers. The authors in [9] proposed a new symmetric encryption for mobile devices. The authors in [10] have introduced a new operative scheme called RoughDorid for detecting malware applications directly on the smartphone. A Dynamic Fully Homomorphic Encryption-based Merkle Tree is proposed in [11]. An outsourced revocation has been introduced in [12] based on identity-based encryption in cloud computing. A new privacy preserving response scheme has been proposed in [13] based on adaptive key evolution in smart grid. A novel dynamic structure for cloud data has been proposed in [14] for efficient public auditing.

Other strategies for applying access control in the cloud via encryption are developed together two research lines: attribute-based encryption (ABE) and discerning encryption procedures. ABE approaches (e.g., [15–18]) supply access control authorities by making sure that the key used to protect

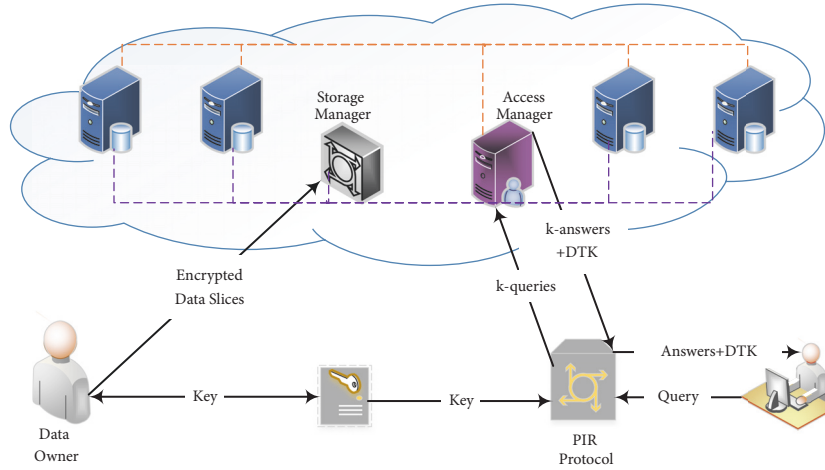


FIGURE 1: Integrated PIR with cloud computing system model.

a source could be derived exclusively by the consumers that meet a specified condition in their characteristics (e.g., era, function). An efficient and secure data outsourcing with check-ability has been proposed in [19] for cloud computing based on ABE. Also, a novel lightweight encryption mechanism for database is introduced in [20]. The authors in [21] introduced a new identity-based signcryption on lattice without trapdoor. For the multiple sources, the authors in [22] proposed a new homomorphic signature scheme based on network coding and applied it to IoT. The privacy is preserved in IoT using centralized duplicate removal video storage system [23]. Also, a new identity-based antequantum blind authentication for privacy preserving in wireless sensor networks has been proposed in [24]. For the blind storage, the authors in [25] have introduced efficient multikeyword ranked search for mobile cloud data. In [26] the authors proposed the personalized search in mobile clouds over encrypted data with efficient updates. Reference [27] proposed a new block design-based key agreement for data sharing in cloud computing.

Procedures based on discerning encryption (e.g., [28, 29]) suppose to encrypt every single source using a secret that only licensed users understand or may derive. Within this situation, the information owner either manages policy upgrades, with overhead, or is assigned to the server. Though overencryption is shown to provide functionality that is decent and promises a prompt authorities of policy upgrades, it needs trust premises that are more powerful on the machine, which has to offer support. Also, another set of contributions [30, 31] have implemented access control for a private cloud computing environment, by proposing the confidence notation for denying or granting access. In the event the host is oblivious of its adoption our scheme may be used. The authors in [32] proposed a flexible EHR sharing scheme supporting offline encryption of EHR and outsourced decryption of EHR ciphertexts in mobile cloud computing. Reference [33] proposed the first lattice-based linearly homomorphic signature in the standard model, which settles this open problem. The authors in [34] proposed a dependable distributed WSN framework for SHM. Then the authors in [35] proposed a new biometrics-based authentication scheme for the multiserver environment.

### 3. PIR with Cloud Computing

The challenge yet is the way to look for an efficient and protected PIR scheme (regarding costs for information storage and recovery). Our proposed scheme is  $k$ -private and  $t$ -out-of- $n$  PIR, which means that if only  $t$  of  $n$  servers is required to respond and even though  $k$  servers collude together, the queried information will not be revealed. Our scheme consists of three basic stages: *Data Storage and Encryption*, *User Authorization and Query*, and *Data Decryption and Reconstruction*. We have employed the basic PIR scheme definition [36].

**3.1. Data Storage and Encryption.** The data  $\mathcal{D}$  will be stored on  $k$  cloud server ( $\mathcal{SRV}_1, \mathcal{SRV}_2, \dots, \mathcal{SRV}_i, \dots, \mathcal{SRV}_k$ ) after encrypting it with our encryption algorithm. We consider three types of servers in the data storage side, as shown in our system model Figure 1. The *Storage Manager Server* receives the encrypted data slices and distributes them on the *Data Storage Servers* under its own control, and the *Key-Server* is responsible for the key management that is distributed using the *PIR Protocol*.

The owner encrypts all the information bits using a content key using symmetric encryption procedures. The content essential is then encrypted by the owner. It requires as inputs the public key handled by the *PIR Protocol*, the content key  $ck$ , and an access structure  $\mathcal{A} = (M, \rho)$ . Let  $M$  be a  $l \times n$  matrix, where  $l$  denotes the total number of all the attributes. The function  $\rho$  associates rows of  $M$  to attributes.

$$\mathcal{D} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_i^1 & \cdots & x_k^1 \\ x_1^2 & x_2^2 & \cdots & x_i^2 & \cdots & x_k^2 \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ x_1^j & x_2^j & \cdots & x_i^j & \cdots & x_k^j \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ x_1^n & x_2^n & \cdots & x_i^n & \cdots & x_k^n \end{bmatrix}$$

$$\begin{aligned}
& \begin{bmatrix} \overline{x}_1^{h(1)} & \overline{x}_2^{h(1)} & \dots & \overline{x}_i^{h(1)} & \dots & \overline{x}_k^{h(1)} \\ \overline{x}_1^{h(2)} & \overline{x}_2^{h(2)} & \dots & \overline{x}_i^{h(2)} & \dots & \overline{x}_k^{h(2)} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ \overline{x}_1^{h(j)} & \overline{x}_2^{h(j)} & \dots & \overline{x}_i^{h(j)} & \dots & \overline{x}_k^{h(j)} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ \overline{x}_1^{h(n)} & \overline{x}_2^{h(n)} & \dots & \overline{x}_i^{h(n)} & \dots & \overline{x}_k^{h(n)} \end{bmatrix} \\
& = [s_1 \ s_2 \ \dots \ s_i \ \dots \ s_k]
\end{aligned} \tag{1}$$

where each element of the vector  $[s_1, s_2, \dots, s_i, \dots, s_k]$  represents the data to be stored on each server  $(\mathcal{SRV}_1, \mathcal{SRV}_2, \dots, \mathcal{SRV}_i, \dots, \mathcal{SRV}_k)$  respectively. Each  $s_i$  is the corresponding column after applying  $h(j)$  on each piece of data.  $h(j)$  is our hash permutation function that is used to hide the index of each piece of the stored data. More precisely, a hash function  $h$  maps bit strings of arbitrary finite length to strings of fixed length, say  $n$  bits. For a domain  $D$  and range  $R$  with  $h : D \rightarrow R$  and  $|D| > |R|$ , the function is many-to-one, implying that the existence of collisions (pairs of inputs with identical output) is unavoidable.

**Definition 1** (Hash Permutation Function  $h$ ). A hash function is a function  $h : D \rightarrow R$  and  $|D| > |R|$  maps bit strings of arbitrary finite length to strings of fixed length, which has the following properties:

- (1) Compression:  $h$  maps an input  $x_i$  of arbitrary finite bit length, to an output  $h(x_i)$  of fixed bit length  $n$ .
- (2) Ease of computation: given  $h$  and an input  $x_i$ ,  $h(x_i)$  is easy to compute.
- (3) Preimage resistance: for essentially all prespecified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage  $x_i$  such that  $h(x_i) = y$  when given any  $y$  for which a corresponding input is not known.
- (4) 2nd-preimage resistance: it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given  $x_j$ , to find a 2nd-preimage  $x_i \neq x_j$  such that  $h(x_i) = h(x_j)$ .
- (5) Collision resistance: it is computationally infeasible to find any two distinct inputs  $x_i, x_j$  which hash to the same output, i.e., such that  $h(x_i) = h(x_j)$ .

We can find that the collision resistance implies 2nd-preimage resistance of hash functions, but collision resistance does not guarantee preimage resistance.

**3.2. User Authorization and Query.** The user can issue a query  $Q$  to receive a file  $x$  which is partitioned and stored on  $k$  different server  $(\mathcal{SRV}_1, \mathcal{SRV}_2, \dots, \mathcal{SRV}_i, \dots, \mathcal{SRV}_k)$ . Considering that the information is hosted on both cloud servers. Then there has to be a decryption token for every user to have the ability to synchronize the information. The

user has to issue a query to the PIR protocol including the user's secret key, attributes, and certificates. The PIR protocol will issue  $k$ -queries (one for each server)  $Q = \{q_1, q_2, \dots, q_i, \dots, q_k\}$  for the access manager server, where  $q_i = Q_i(k, n, j)$  and  $j$  is a randomly chosen by flipping coins. Each server will respond with a single encrypted answer; the user will have an encrypted answer set  $EA = \{ea_1, ea_2, \dots, ea_i, \dots, ea_k\}$ , where  $ea_i = EA_i(k, i, x, q_i)$ . The access manager server will reply with the servers' answers and the decryption token that is sent to the user. Then, it will have the ability to decrypt and rebuild the requested information.

**3.2.1. Decryption Token Generation.** The decryption token generation algorithm (Algorithm 2) is run by the access manager server. It requires as inputs the ciphertext  $CT$  that implicitly includes an access structure  $\mathcal{A}$ , user's public key  $UPK$  generated by the PIR protocol, user's secret key  $USK$ , and user's set of attributes  $USA$ . When  $USA$  suits the accessibility construction  $\mathcal{A}$ , the algorithm could successfully calculate the right decryption token  $DTK$  of this ciphertext. Thus, the PIR protocol can transform the user's query to a set of  $k$ -queries for the *Access Manager*.

**3.3. Data Decryption and Reconstruction.** Once the user has received the encrypted answers set  $EA$  and its decryption token  $DTK$ , he can decrypt the answers with the help of its own secret key  $USK$  and get the answer set  $A$  that will be used for reconstruction. Since the permutation function can make some collisions ( $NC$ ) probability  $p$ , the success probability is  $1 - (1 - p)^k$ , and the reconstruction threshold  $t = k - NC$ . Based on the value of  $t$  the user will be able to reconstruct the requested slice. Once the data has been reconstructed, the user can use its given decryption token to decrypt the data.

## 4. Implementation

Our model is implemented on the top of our private cloud environment, which is based OpenStack [37]. We have built our proposed scheme by initializing a set of virtual instances that are divided into three categories based on their configuration:

- (i) **Category 1:**  $n$  virtual instances working as storage servers. The configuration of those  $n$  servers is 4 VCPUS, 4 GB RAM, and 80 GB disk. We have considered  $n = 50$ ; thus, the IP addresses for those servers are 10.0.10.101 : 10.0.10.150 with subnet mask 255.255.255.0. Those storage servers have two basic tasks, storing the encrypted data slices submitted to them through the *Storage Manager*. The second task is sending the encrypted data slices back as answers to the *Access Manager* server to be delivered to the requesting user.
- (ii) **Category 2:** two virtual instances. The configuration for each of them is 4 VCPUS, 8 GB RAM, and 20 GB disk. They are working as *PIR Protocol* server with IP address 10.0.10.151 and *Key Manager* server with IP address 10.0.10.152. After the  $k - answers$  are received by the *PIR Protocol* from the *Access Manager*

<b>Input:</b>	(i) $ck_i$ <span style="float: right;">▷ The content key for each <math>x_i</math></span> (ii) $PK$ <span style="float: right;">▷ The data <math>\mathcal{D}</math> public key managed by Shamir secret sharing</span> (iii) $\mathcal{A} = (M, \rho)$ <span style="float: right;">▷ The data <math>\mathcal{D}</math> access structure, which will be implicitly included in the ciphertext</span>
	(1) Choose $\alpha_i, \beta_i, \gamma_i \in \mathbb{Z}_q^*$ <span style="float: right;">▷ The random secrets of each data slice <math>x_i</math></span> (2) Choose $f \in \mathbb{Z}_q^*$ <span style="float: right;">▷ A random encryption exponent</span> (3) Choose $\vec{v} = [f, y_2, \dots, y_n]^T \in \mathbb{Z}_q^{*n}$ <span style="float: right;">▷ A random vector, where <math>[y_2, \dots, y_n]</math> are used to share the random encryption exponent <math>f</math></span> (4) <b>for</b> $j = 1$ to $l$ <b>do</b> (5)     Compute $\lambda_j = \vec{v} \cdot M_j$ <span style="float: right;">▷ <math>M_j</math> is the vector corresponding to the <math>j</math>-th row of the matrix <math>M</math></span> (6) <b>end for</b> (7) Randomly choose $r_1, \dots, r_l \in \mathbb{Z}_q^*$ (8) Compute: $C' = g^f$ and $C'' = g^{f/\beta_i}$ (9) <b>for</b> $i = 1$ to $l$ <b>do</b> (10)     Compute $C_j = g^{\alpha_j} \cdot ((g^{\vec{v}(j)\rho(j)} H(\rho(j))^{y_i})^{-r_j})$ (11)     Compute $D_{1,j} = g^{r_j/\beta_i}$ and $D_{2,j} = g^{-(y_i/\beta_i)r_i}$ (12) <b>end for</b> (13) Compute the ciphertext:
	$CT_i = \left[ ck_i, \left( \prod_j^l \hat{e}(g, g)^{\alpha_i} \right)^f, C', C'', C_j, D_{1,j}, D_{2,j}, \rho(j) \right]$
<b>Output:</b>	The ciphertext $CT$

ALGORITHM 1: Encryption.

server, the user must not be given so much data slices to guarantee the data secrecy. Also, the user should not be given less data slices than the required slices that can generate the requested data. Thus, the *PIR Protocol* server runs the PIR protocol to ensure hiding the requested data slices identity from the cloud storage servers and also ensuring granting the user the appropriate data slices to be able to recover the requested data. The *Key Manager* server is responsible for assigning the keys for both the data owners and requesting users.

- (iii) **Category 3:** two virtual instances. The configuration for each of them is 8 VCPUS, 16 GB RAM, and 40 GB disk. They are working as *Storage Manager* server with IP address 10.0.10.154 and *Access Manager* server with IP address 10.0.10.155. The *Storage Manager* receives the encrypted data slices and distributes them on the appropriate cloud storage instances. The *Access Manager* receives the encrypted data slices from the cloud storage instances and sends them to the *PIR Protocol*.

It should be mentioned that all of those virtual instances are cooperating together to construct the proposed scheme. Each of them has its own task that can perfectly execute it.

## 5. Performance Analysis

The performance analysis of our proposed scheme is introduced based on two mandatory scenarios.

**5.1. Client Situation.** Our scheme requires the user to perform a more intricate decryption compared to using Attribute Encryption Scheme (AES) using a conventional encryption manner, by introducing the Token Generation Algorithm 2. In our scheme the decryption is parallelized on several core VCPUS, which makes the user processing much more effective. In our experiments, we have considered five different size categories: *32:127 bits*; *128:511 bits*; *512:2047 bits*; *2048:8191 bits*; and *8192:32768 bits*.

Figure 2 shows that the decryption cost can be used with all acceptable scenarios for the use of our scheme. Specifically, the figure illustrates the throughput obtained by changing the number of slices, through implementing our scheme in various configurations defined by five size groups (*32:127 bits*, *128:511 bits*, *512:2047 bits*, *2048:8191 bits*, and *8192:32768 bits*). Based on the execution of our encryption protocol (Algorithm 1), we notice that even the largest size category (*8192:32768 bits*) offers a throughput that is approximately 85 MB/s, while considering 16 slices of that size category. The throughput for the smallest size category (*32:127 bits*) is about 140 MB/s, while considering 16 slices of that size category. The figure also reveals that decreasing the amount of slices, we achieve the performance level that is 1.5 times that obtained from the *8192:32768 bits* size group and 2 times the one obtained by *32:127 bits* size group. It should be noted that the experimental results for that part are the average of 25 trails.

**5.2. Overlay Situation.** In our proposed scheme, the overlay situation is analyzed by using the Swift (it organizes objects

**Input:** (i)  $CT_x$   $\triangleright$  The ciphertext related to the file  $x$   
(ii)  $UPK$   $\triangleright$  The user's public key given by PIR protocol  
(iii)  $USK$   $\triangleright$  The user's secret key  
(iv)  $UA$   $\triangleright$  The user's set of attributes

(1) Let:  $CT_A = |\mathcal{A}|$   $\triangleright$  The set of attributes involved in  $CT_x$   
(2) Choose a set of constants  $w_i \in \mathbb{Z}_q^*$ ,  $\forall i \in CT_A$   
(3) **for**  $i = 1$  to  $CT_A$  **do**  
(4) **if**  $\lambda_i \in Share(f)$  **then**  $\triangleright \lambda_i$  are valid shares of the secret  $f$  according to the data  $\mathcal{D}$   
(5) Reconstruct the encryption exponent:  $f = \sum_{i=1}^{CT_A} w_i \lambda_i$   
(6) **end if**  
(7) **end for**  
(8) Let  $\{R_{U,i}, K_{U,i}, L_{U,i} \in \mathbb{Z}_q^* \mid \forall i \in CT_A\}$  are random constants  
(9) **if**  $UA \models \mathcal{A}$  **then**  $\triangleright$  The user's attributes satisfies  $\mathcal{A}$

$$DTK = \prod_{i=1}^{CT_A} \frac{\hat{e}(C', K_{U,i}) \cdot \hat{e}(C'', R_{U,i})^{-1}}{\prod_{i=1}^{CT_A} [\hat{e}(C_i, UPK_i) \cdot \hat{e}(D_{1,i}, K_{U,\rho(i)}) \cdot \hat{e}(D_{2,i}, L_{U,i})]^{w_i CT_A}}$$

$$= \frac{\hat{e}(g, g)^{a.U.f.CT_A} \cdot \prod_{i=1}^{CT_A} \hat{e}(g, g)^{(\alpha_i/USK)f}}{\hat{e}(g, g)^{a.U.f.CT_A} \cdot \sum_{i=1}^{CT_A} w_i \lambda_i}$$

$$= \prod_{i=1}^{CT_A} \hat{e}(g, g)^{(\alpha_i/USK)f}$$

(10) **else**  $\triangleright DTK$  cannot successfully computed  
(11)  $DTK = \text{rand}(\text{Hex})$   $\triangleright DTK$  will get a random Hexadecimal value  
(12) **end if**  
**Output:** The Decryption Token  $DTK$

ALGORITHM 2: Token generation.

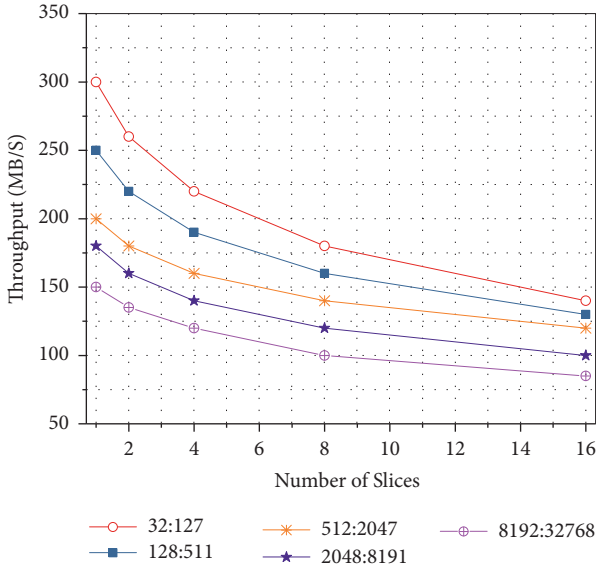


FIGURE 2: Throughput varying the number of considered slices for four different size categories.

within containers) service as a reference. We have adopted the DLO support offered by Swift to implement the `get` and `put_fragment` methods that characterize our scheme.

With our experiments at the overlay scenario, we have assembled a Swift user program in Python that implements the `get` and also `put_fragment` techniques that describe our strategy. We have followed two strategies: one is by fragmenting an object as atomic objects; the second is by using DLO introduced by Swift.

Figure 3 contrasts the period required for the implementation of `get` requests based on different amounts of contemplated fragments (1, 4, 16, 64, 256, and 1024) to a specific object. The operation is proved dependent on the network bandwidth and also the overhead imposed by the management of every `get` request.

Specifically for `get` requests, the overhead introduced into handling one portion for every fragment predominates in the event of little costs, whereas the growth in object's size exhausts the system's bandwidth that causes a great bottleneck. By contemplating an item of size 1GB, the time required for implementing the `get` requests is roughly 92 seconds for 1 considered fragment, and the time is 1000 seconds for 1024 considered fragments for the same object size. In case of considering an object of size 64 KB, the time required for executing the `get` requests is about 0.08 seconds for 1 considered fragment, and the time is 50 seconds for 1024 considered fragments for the same object size.

Figure 4 reports the throughput for a workload caused by the implementation of blending `get` and also `put_fragment` requests on Swift by changing the object size and the amount

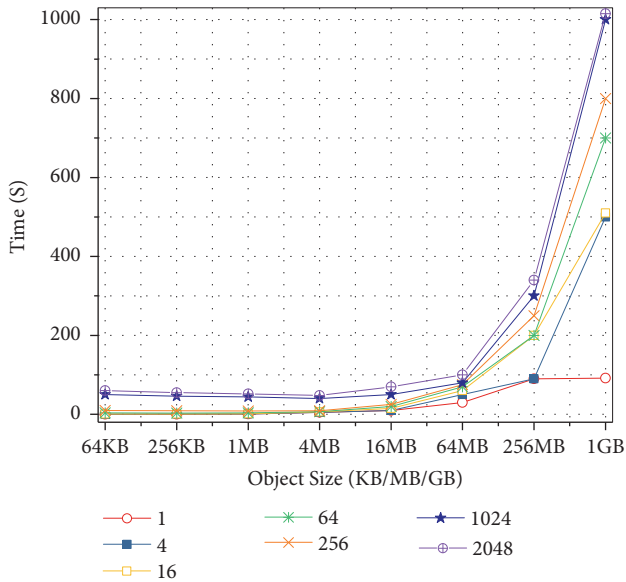


FIGURE 3: The execution time for the `get` requests on Swift.

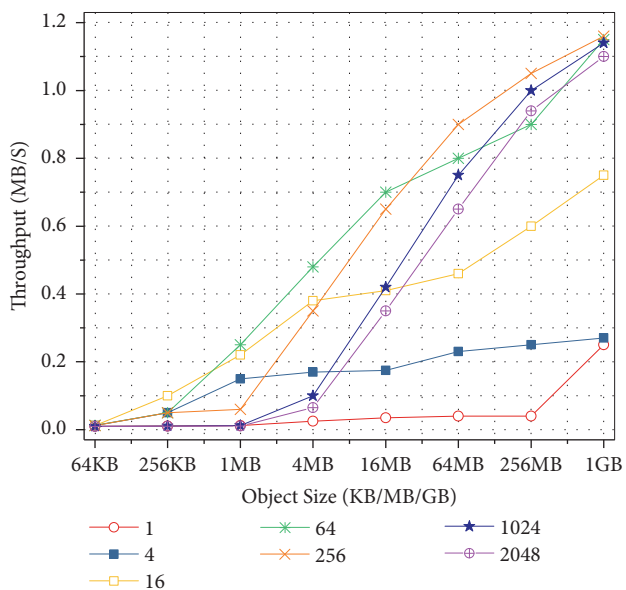


FIGURE 4: The throughput for a workload introduced by combining the `get` and `put_fragment` requests on Swift.

of contemplated fragments for the exact same object. We have assessed the behavior of our scheme based on a selection of 2048 objects, where following every `put_fragment` request, a succession of 100 `get` requests were implemented on objects in precisely the exact same group and are all of the exact same size. These configurations using fragments' throughput will be orders of magnitude greater already. The figure also demonstrates that the very best number of fragments is dependent upon the resource size. The identification of this value needs to think about the setup of the workload and this machine.

## 6. Conclusion

We presented a robust PIR scheme for efficiently and securely storing and retrieving private information from untrusted cloud servers. Our scheme lets the data owners to effectively divide and encrypt their own data into small slices of five different size categories. Our implementation and experimental analysis confirm the efficacy and efficacy of our proposed scheme, which appreciates orders of magnitude of improvement in throughput concerning source protection and decryption. For an object of size 1 GB, the time required for implementing the `get` requests is roughly 92 seconds for 1 considered fragment; the time is 1000 seconds for 1024 considered fragments for the same object size. Considering an object of size 64 KB, the time required for executing the `get` requests is about 0.08 seconds for 1 considered fragment, and the time is 50 seconds for 1024 considered fragments for the same object size. The proposed scheme also supports its compatibility with all present cloud storage environments, which makes it also relevant to a lot of application domains.

## Data Availability

The data used to support the findings of this study are available from the authors upon request.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## References

- [1] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proceedings of the 36th Annual Foundations of Computer Science*, pp. 41–50, Los Alamitos, Calif, USA, October 1995.
- [2] B. Chor and N. Gilboa, "Computationally private information retrieval (extended abstract)," in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97)*, pp. 304–313, New York, NY, USA, May 1997.
- [3] R. L. Rivest, "All-or-nothing encryption and the package transform," in *Fast Software Encryption*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 210–218, Springer Berlin Heidelberg, 1997.
- [4] J. Shen, C. Wang, T. Li, X. Chen, X. Huang, and Z.-H. Zhan, "Secure data uploading scheme for a smart home system," *Information Sciences*, vol. 453, pp. 186–197, 2018.
- [5] X. Chen, J. Li, J. Weng, J. Ma, and W. Lou, "Verifiable computation over large database with incremental updates," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 65, no. 10, pp. 3184–3195, 2016.
- [6] X. Chen, J. Li, X. Huang, J. Ma, and W. Lou, "New Publicly Verifiable Databases with Efficient Updates," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 546–556, 2015.
- [7] H. Sun and S. A. Jafar, "The capacity of robust private information retrieval with colluding databases," *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, vol. 64, no. 4, part 1, pp. 2361–2370, 2018.

- [8] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, "Protecting data privacy in private information retrieval schemes," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 592–629, 2000.
- [9] C. Gao, S. Lv, Y. Wei, Z. Wang, Z. Liu, and X. Cheng, "M-SSE: an effective searchable symmetric encryption with enhanced security for mobile devices," *IEEE Access*, vol. 6, pp. 38860–38869, 2018.
- [10] K. Riad and L. Ke, "Operative scheme for functional android malware detection," *Security and Communication Networks*.
- [11] J. Xu, L. Wei, Y. Zhang, A. Wang, F. Zhou, and C. Gao, "Dynamic Fully Homomorphic encryption-based Merkle Tree for lightweight streaming authenticated data structures," *Journal of Network and Computer Applications*, vol. 107, pp. 113–124, 2018.
- [12] J. Li, X. Chen, C. Jia, and W. Lou, "Identity-based encryption with outsourced revocation in cloud computing," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 425–437, 2015.
- [13] H. Li, X. Lin, H. Yang, X. Liang, R. Lu, and X. Shen, "EPPDR: an efficient privacy-preserving demand response scheme with adaptive key evolution in smart grid," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2053–2064, 2014.
- [14] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2402–2415, 2017.
- [15] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*, pp. 89–98, November 2006.
- [16] K. Riad, "Multi-authority trust access control for cloud storage," in *Proceedings of the 4th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS '16)*, pp. 429–433, August 2016.
- [17] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, pp. 1214–1221, 2011.
- [18] K. Riad, "Revocation basis and proofs access control for cloud storage multi-authority systems," in *Proceedings of the 3rd International Conference on Artificial Intelligence and Pattern Recognition (AIPR '16)*, pp. 118–127, September 2016.
- [19] J. Li, X. Y. Huang, J. W. Li, X. F. Chen, and Y. Xiang, "Securely outsourcing attribute-based encryption with checkability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2201–2210, 2014.
- [20] J. Li, Z. Liu, X. Chen, F. Khafa, X. Tan, and D. S. Wong, "L-EncDB: A lightweight framework for privacy-preserving data queries in cloud computing," *Knowledge-Based Systems*, vol. 79, pp. 18–26, 2015.
- [21] X. W. Y. Zhang, H. Zhu, and L. Jiang, "An identity-based signcryption on lattice without trapdoor," *Journal of Universal Computer Science*, 2018.
- [22] T. Li, W. Chen, Y. Tang, and H. Yan, "A homomorphic network coding signature scheme for multiple sources and its application in IoT," *Security and Communication Networks*, vol. 2018, Article ID 9641273, 6 pages, 2018.
- [23] H. Yan, X. Li, Y. Wang, and C. Jia, "Centralized duplicate removal video storage system with privacy preservation in IoT," *Sensors*, vol. 18, no. 6, 2018.
- [24] H. Zhu, Y. Tan, L. Zhu, X. Wang, Q. Zhang, and Y. Li, "An identity-based anti-quantum privacy-preserving blind authentication in wireless sensor networks," *Sensors*, vol. 18, no. 5, 2018.
- [25] H. Li, D. Liu, Y. Dai, T. H. Luan, and X. S. Shen, "Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 1, pp. 127–139, 2015.
- [26] H. Li, D. Liu, Y. Dai, T. H. Luan, and S. Yu, "Personalized search over encrypted data with efficient and secure updates in mobile clouds," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 1, pp. 97–109, 2018.
- [27] J. Shen, T. Zhou, D. He, Y. Zhang, X. Sun, and Y. Xiang, "Block design-based key agreement for group data sharing in cloud computing," *IEEE Transactions on Dependable and Secure Computing*, 2018.
- [28] S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced data," *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 2, 2010.
- [29] I. Hang, F. Kerschbaum, and E. Damiani, "ENKI: Access control for encrypted query processing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*, pp. 183–196, New York, NY, USA, June 2015.
- [30] K. Riad, "Blacklisting and forgiving coarse-grained access control for cloud computing," *International Journal of Security and Its Applications*, vol. 10, no. 11, pp. 187–200, 2016.
- [31] K. Riad and Z. Yan, "Multi-factor synthesis decision-making for trust-based access control on cloud," *International Journal of Cooperative Information Systems*, vol. 26, no. 04, pp. 1–33, 2017.
- [32] Z. Cai, H. Yan, P. Li, Z.-A. Huang, and C. Gao, "Towards secure and flexible EHR sharing in mobile health cloud under static assumptions," *Cluster Computing*, vol. 20, no. 3, pp. 2415–2422, 2017.
- [33] W. Chen, H. Lei, and K. Qi, "Lattice-based linearly homomorphic signatures in the standard model," *Theoretical Computer Science*, vol. 634, pp. 47–54, 2016.
- [34] M. Z. A. Bhuiyan, G. Wang, J. Wu, J. Cao, X. Liu, and T. Wang, "Dependable structural health monitoring using wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 4, pp. 363–376, 2017.
- [35] H. Shen, C. Gao, D. He, and L. Wu, "New biometrics-based authentication scheme for multi-server environment in critical systems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 6, pp. 825–834, 2015.
- [36] A. Fazeli, A. Vardy, and E. Yaakobi, "PIR with Low Storage Overhead: Coding Instead of Replication," 2015, <https://arxiv.org/abs/1505.06241>.
- [37] OpenStack, <http://www.openstack.org/>.





**Hindawi**

Submit your manuscripts at  
[www.hindawi.com](http://www.hindawi.com)

